AD-A208 019

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

DTIC FILE COPY

4

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AIM 1006	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Task-Level Robot Learning: Ball Throwing		5. TYPE OF REPORT & PERIOD COVERED memorandum
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Eric Aboaf, Christopher G. Atkeson and David J. Reinkensmeyer		8. CONTRACT OR GRANT NUMBER(s) N00014-86-K-0685 N00014-85-K-0124
9. PERFORMING ORGANIZATION NAME AND ADDRESS Artificial Intelligence Laboratory 545 Technology Square Cambridge, MA 02139		10. PROGRAM ELEMENT PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Advanced Research Projects Agency 1400 Wilson Blvd. Arlington, VA 22209		12. REPORT DATE December 1987
		13. NUMBER OF PAGES 18
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Office of Naval Research Information Systems Arlington, VA 22217		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Distribution is unlimited		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES None		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) robotics learning tasks		

DTIC
ELECTE
MAY 22 1989
E

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Abstract. We are investigating how to program robots so that they learn tasks from practice. One method, *task-level learning*, provides advantages over simply perfecting models of the robot's lower level systems. Task-level learning can compensate for the structural modeling errors of the robot's lower level control systems and can speed up the learning process by reducing the degrees of freedom of the models to be learned. We demonstrate two general learning procedures—fixed-model learning and

See back

89 5 22

Block 20 continued.

refined-model learning—on a ball-throwing robot system. Both learning approaches refine the task command based on the performance error of the system, while they ignore the intermediate variables separating the lower level systems. We provide both experimental and theoretical evidence that task-level learning can improve a robot's performance of a task. ([illegible]

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

QUALITY INSPECTED 2

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo 1006 December 1987

TASK-LEVEL ROBOT LEARNING: BALL THROWING

Eric W. Aboaf
Christopher G. Atkeson
David J. Reinkensmeyer

Abstract. We are investigating how to program robots so that they learn tasks from practice. One method, *task-level learning*, provides advantages over simply perfecting models of the robot's lower level systems. Task-level learning can compensate for the structural modeling errors of the robot's lower level control systems and can speed up the learning process by reducing the degrees of freedom of the models to be learned. We demonstrate two general learning procedures—fixed-model learning and refined-model learning—on a ball-throwing robot system. Both learning approaches refine the task command based on the performance error of the system, while they ignore the intermediate variables separating the lower level systems. We provide both experimental and theoretical evidence that task-level learning can improve a robot's performance of a task.

Acknowledgments. This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the A. I. Laboratory's research is provided in part by the Office of Naval Research University Research Initiative Program under Office of Naval Research contract N00014-86-K-0685, and the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124. Partial support for E. Aboaf was provided by an Office of Naval Research Graduate Student Fellowship. Partial support for C. Atkeson was provided by a National Science Foundation Engineering Initiation Award and Presidential Young Investigator Award.

© Massachusetts Institute of Technology 1987

1 Introduction

We are investigating how to improve a robot's performance of a task. One popular approach is to calibrate the models that describe a robot's lower level systems—dynamics, kinematics, and perception. We are examining a different approach in which the robot system as a whole practices the desired task and learns from its experience. After each attempt at the task, a learning procedure is used to interpret the errors in task performance in order to adjust the commands that drive the robot.

Task-level learning can proceed when the robot practices the task. The robot performs a task using its control system to generate appropriate commands, and then monitors its own performance with its perceptual apparatus. After each attempt at the task, a learning robot can use a model of the task to interpret the error in performance. With this information the robot can modify its original commands in order to better achieve the task on the next trial. The robot then performs the task again with a refined command, monitors the new results, and attempts the task once again if the performance goals are still not satisfied. The robot continues this sequence of performing the task, monitoring performance, and refining commands until the task is achieved.

We have demonstrated this process of task-level learning with a ball-throwing robot system that improves its performance with practice (Figure 1). Given an estimate of the target, the system uses a ballistic model, a kinematic model, and a dynamics model to calculate a desired trajectory for the robot arm. The robot then throws the ball at the target. A vision system measures where the ball lands with respect to the target. Based on the error in performance, the system applies a task-level learning procedure to modify the desired trajectory of the robot arm. The robot system then throws the ball again with this updated trajectory.

The task-level learning procedure modifies the system command based on the error in task performance. We have developed two learning algorithms—fixed model learning and refined model learning—that were implemented on the ball-throwing robot system. Fixed-model learning modifies the system command by applying an inverse model of the system to the performance error. Refined-model learning is an interpolation procedure that refines the system command and improves the inverse model.

Applying learning at the task level is a promising approach for making robots perform a wide range of common tasks. Learning provides a method of compensating for the structural modeling errors of the robot's lower level systems. Learning can also be applied at the task level to improve the performance of the entire robot system without focusing on each lower level system. In this paper we describe two task-level learning algorithms, demonstrate their effectiveness on a ball-throwing task, and

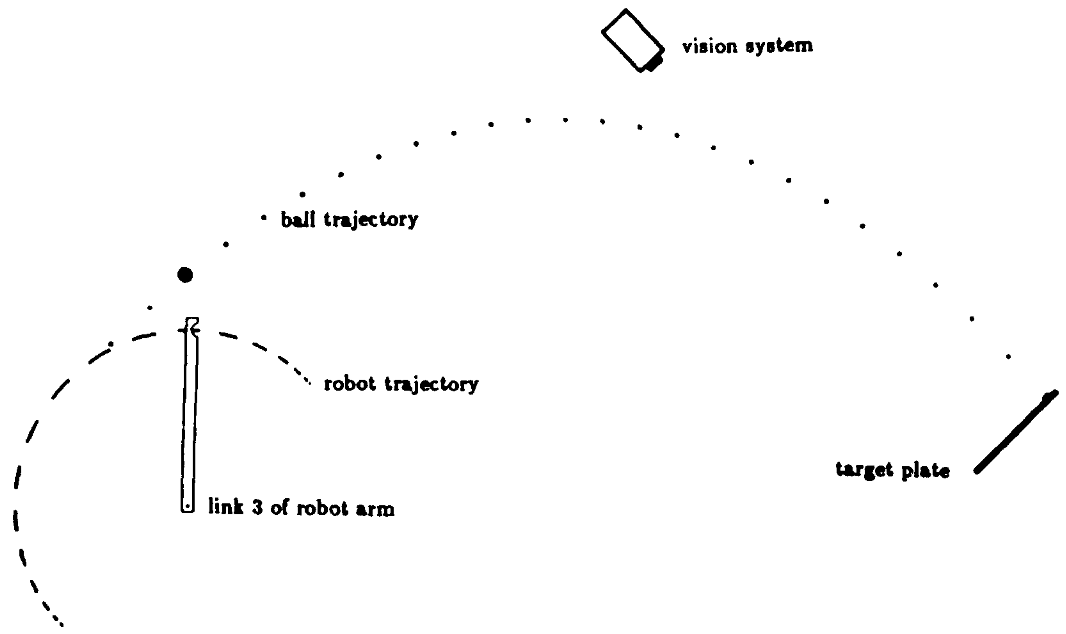

Figure 1: Ball-Throwing Robot System

discuss the issues involved in applying these learning principles to other dynamic tasks.

1.1 Modeling and Learning

The robot system performs the task of throwing a ball at the target. The ball-throwing robot can be considered an input/output system (Figure 2) driven by input variables ($\mathbf{c}$) and responding with outputs variables ($\mathbf{p}$)

$$\mathbf{p} = \mathbf{S}(\mathbf{c}) \tag{1}$$

The system is denoted by $\mathbf{S}()$, the system inputs are termed *commands*, and the system outputs are termed *performance.* Figure 2 also shows that the lower level modules of the ball-throwing system—ballistics, kinematics, and dynamics—are concatenated to form the system $\mathbf{S}()$.

An *inverse model* of the system is used to choose commands $\mathbf{c}$ that will yield the desired task performance $\mathbf{p_d}$

$$\mathbf{c} = \hat{\mathbf{S}}^{-1}(\mathbf{p_d}) \tag{2}$$

A carct ($\hat{}$) denotes the *model* of the system. Figure 3 shows the block diagram of the inverse model, as well as the block diagram that includes the lower level modules.

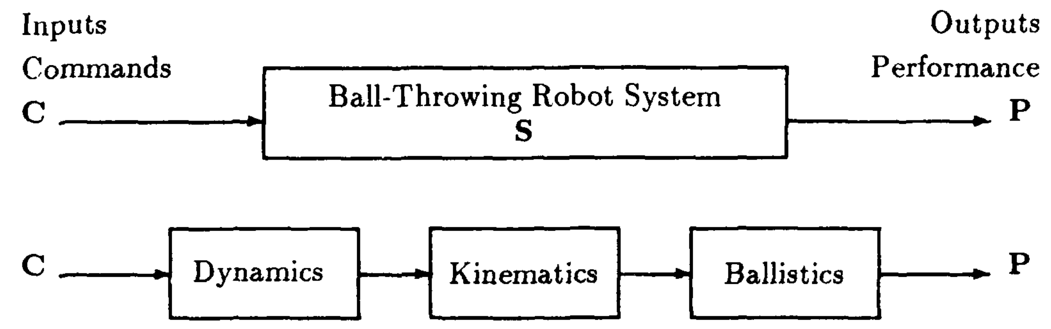

Figure 2: System Description

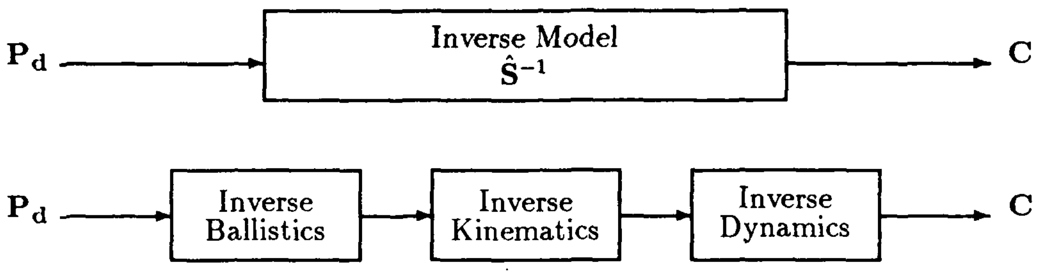

Figure 3: Inverse Model

Driving the ball-throwing system and the ball throwing model with the same command **c** will generally cause different results. The command **c** to the system will produce the actual performance which we label **p**, while the command **c** to the model will produce the desired performance $\mathbf{p}_d$ (Figure 4). The difference between actual performance and desired performance is termed the performance error. The performance error is not zero because the model only approximates the actual system.

The performance error can be reduced either by improving the model of the system or by refining the command **c**. With a better model, the discrepancy between actual and desired performance would be smaller. With a learning procedure, the performance error can be used to refine the command **c**, thereby improving performance on successive iterations. Using both approaches *together* will more effectively improve performance.

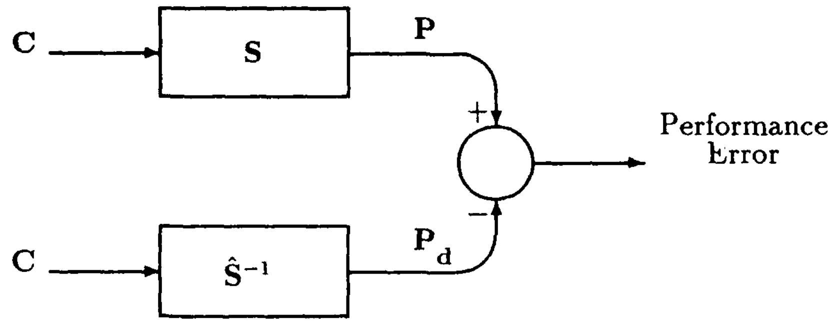

Figure 4: Performance Error

1.2 Relation to Previous Work

We will briefly survey the fields of robot calibration and trajectory learning as they apply to robot learning and robot functionality. Calibration research focuses on modeling robots more accurately so that they can perform a wider range of tasks. Trajectory learning research emphasizes learning from practice as a way to improve trajectory following performance.

One way to increase robot functionality is to control the robot using better models. A good deal of research in robotics has thus been directed towards accurately modeling the lower level modules of robot systems. With these calibrated models, researchers are able to command robots to perform a wider range of tasks. We will thus discuss the advantages and disadvantages of calibration as a way to increase robot functionality. In particular, we will compare the calibration approach to learning from practice.

Much recent work has concentrated on accurately calibrating the kinematic and dynamic models of robots. In kinematic calibration, for example, the actual Denavit-Hartenberg parameters are estimated based on experimental motions of the robot [Hayati and Mirmirani 1985; Wu 1984]. This kinematic model is often extended (and complicated) to include many non-linear effects that the Denavit-Hartenberg model neglects [Whitney et al. 1986]. Much work has also been done in the area of dynamic calibration, where the the actual masses and inertias of the links and loads are estimated based on trial motions of the robot [Atkeson et al. 1986; Mayeda et al. 1984; Mukerjee 1984; Neuman and Khosia 1985; Olsen and Bekey 1985].

Accurately modeling the lower level modules of a robot system has several advantages. When the structure of the model is chosen correctly and the parameters are estimated correctly, the model is valid for any inputs and outputs. The experience gained in selected trials generalizes to the entire range of operation. As a result, learning does not need to take place every time the model is used. Additionally, a structured model provides a compact method of representing the input/output behavior of a system. Data need not be stored for every potential scenario, but the model can instead be evaluated when necessary.

Attempts at improving models of the robot at the expense of introducing learning procedures also have several shortcomings. First, the number and range of robot motions necessary to fully estimate the model parameters is often large. Making trial motions that actually attempt the task may be a more efficient method of reaching the task goal. Second, no matter how well the parameters are estimated, the models are often based on structural assumptions that are not valid in practice. The Newton-Euler model of dynamics, for example, is based on rigid body dynamics which typically is a good but not perfect description of robots. Compensating for the structural assumptions of the model requires a great deal of data, time, and ingenuity. Even after compensation some structural modeling errors remain.

Robot learning research has focused in the past on the trajectory following subtask [Arimoto et al. 1985; Casalino and Gambardella 1986; Craig 1984; Furuta and Yamakita 1986; Hara et al. 1985; Harokopos 1986; Mita and Kato 1985; Morita 1986, Togai and Yamano 1986; Uchiyama 1978; Wang 1984; Wang and Horowitz 1985]. Robots are made to follow a particular trajectory better as they repeat the movement. Feedforward torque commands for repetitive movements are refined on the basis of previous movement errors. This research has focused primarily on linear learning operators which often ignore the underlying model of the system. The work has also emphasized the stability and not the performance of the proposed algorithms.

We will explore the advantages of using the inverse model as the learning operator and how to apply learning algorithms at the task level. Atkeson and McIntyre [1986] began to explore fixed-model learning for the trajectory following subtask. The research shows that using the inverse Newton-Euler model as the learning operator reduces most of the movement errors. The theoretical convergence criteria and performance for the learning algorithm was then derived [Atkeson et al. 1987]. The success of the algorithm at the task level can now be assessed. The learning algorithm can be applied beyond a single lower level module—the dynamics component learned in trajectory following—to a series of lower level systems that together perform a task. The theoretical predictions of convergence and performance at the task level can then be confirmed experimentally.

2 The Ball Throwing Task

The task is to throw a ball at a target. Figure 1 illustrates the robot system that is configured to accomplish this task. The system includes the MIT Serial Link Direct Drive Arm [Asada and Youcef-Toumi 1987], a target plate, and a video camera. The height of the ball when it hits the target plate is monitored and improved by learning.

The last link of the robot is used to throw the ball at the target plate. The robot is positioned so that the last link of the arm rotates in a vertical plane. The robot swings this link through a 180° arc from 225° to 45°, catapulting the ball to the target. The joint is servoed to a fifth order polynomial trajectory that accelerates the arm from rest until it reaches 135° and then decelerates the arm to rest at 45°. A 4*cm* rubber ball is placed into a 3.5*cm* diameter hole at the end of the third link of the robot. The ball leaves the hole as the robot arm decelerates during the throw. No release mechanism is used, but the release position is assumed to be when the last link is at 135°. The ball is thrown further by shortening the duration of the throw, which increases the release velocity (the trajectory length is held constant). This trajectory duration is the system command **c** that is refined by the learning procedure to achieve the task.

A video camera records where the ball hits the target plate. The impact of the ball is sensed by a force sensor on which the target plate is mounted. This signal is used to choose the video frames to be stored for later analysis. After the throw, the location of the ball on the target plate is manually measured from the appropriate video frame. This location serves to measure the system performance **p** in accomplishing the task.

The initial trajectory duration command and the arm trajectory itself are calculated based on the measured distance between robot and target. A simple ballistic model, including only gravity, is used to represent the flight of the ball

$$y_d = v\sin(\theta_{rel}) \cdot t - \frac{1}{2}g \cdot t^2 \tag{3}$$

$$x_d = v\cos(\theta_{rel}) \cdot t \tag{4}$$

For a given position of the target (x_d, y_d) and release angle θ_{rel}, the necessary release velocity is calculated by eliminating the variable t from Equations (3) and (4). The release angle (assumed to be at 135° so that $\theta_{rel} = 45°$) remains fixed through the experiment as part of the task strategy. Based on the desired release velocity and on the kinematics of the third link, a fifth order polynomial trajectory is computed for the arm that releases the ball with the appropriate velocity.

Feedforward torques and a feedback controller drive the robot arm so that it follows the desired trajectory. The feedforward torques are calculated using the acceleration profile of the trajectory and the estimated inertia of the third link. A position-derivative feedback controller insures that the arm closely follows the desired trajectory on each throw. The control law that includes both feedforward and feedback torques is

$$T = T_{ffwd} - K_p \cdot (\theta - \theta_d) - K_v \cdot (\dot{\theta} - \dot{\theta}_d) \tag{5}$$

where K_p and K_v are the position and velocity feedback gains.

The calculation of the initial trajectory duration has been based on an *inverse model* of the system. Given the distance and height between robot and target, the trajectory duration is computed. This model is only an approximation to the real system, and thus when the ball is initially thrown the actual performance usually differs from the desired performance. Many factors can cause this performance error including an inaccurate measurement of target location, inaccurate kinematic model, inaccurate vision system, inaccurate feedfoward torques, air resistance, torque saturation, arm command miscalibration, sensor miscalibration, and sensor noise, as well as errors in release angle, release velocity, actual trajectory duration, and others.

The basis of task-level learning in this throwing experiment is to modify the top-level system command so that task performance improves. We present two general learning methods that improve system performance with practice. They both use the task performance error as well as the system model to refine the system command.

2.1 Fixed-Model Learning

In fixed-model learning the inverse model is used to estimate the correct commands based on the performance errors. A correction term, which we label Δ, is updated after each attempt at the task. In ball throwing, the error correction is updated after each throw by the amount the ball missed the target along the target plane. This measured error—whether positive or negative—updates Δ as a running sum

$$\Delta_{n+1} = \Delta_n - (\mathrm{hit}_n - \mathrm{target}) \tag{6}$$

This error correction is added to the desired performance and transformed through the inverse model to calculate the new trajectory duration

$$\mathrm{duration}_{n+1} = \hat{S}^{-1}(\mathrm{target} + \Delta_{n+1}) \tag{7}$$

A physical interpretation of Equation (7) is to consider the quantity (target + Δ_{n+1}) to be the aim. In the case where the ball falls short, the error correction is positive, raising the aim by the amount of performance error. This action corresponds to our

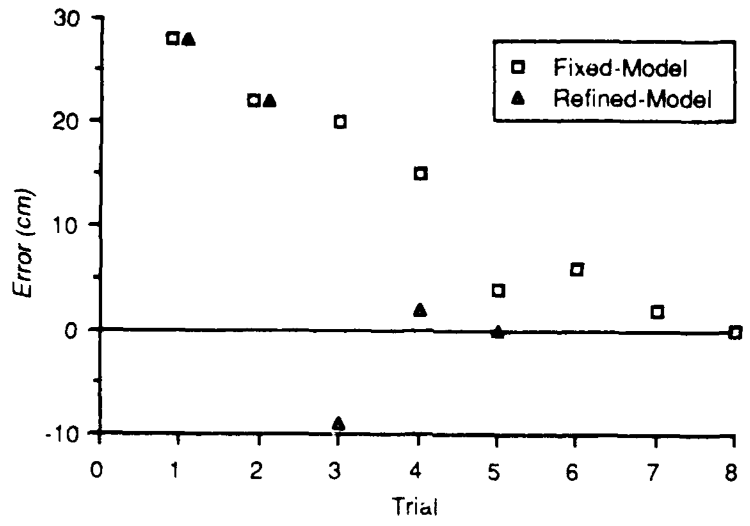

Figure 5: Learning Performance

intuition that we should aim higher if we are hitting too low. Together, Equations (6) and (7) provide the basis for fixed-model learning.

Fixed-model learning was applied to the ball throwing task. The target was placed at a horizontal distance of 5.75 meters and a height of -0.9 meters from the robot. Using the ballistic and kinematic models of the robot system, a commanded trajectory duration of 138 milliseconds was calculated with Equation (7). (Δ_0 is defined to be zero.) The robot threw the ball with this command, resulting in a performance error of $28cm$ in the target plane. The error correction term was then calculated to be $-28cm$ using Equation (6), and the commanded trajectory duration was refined to $143ms$ using Equation (7). On the next throw, the performance error was $22cm$, and the error correction term was calculated to be $-50cm$. This iterative learning procedure was followed until the robot successfully completed the task. The trajectory duration for this successful throw was $156ms$; the error correction was $-97cm$. The open boxes in Figure 5 show the performance improvement with practice. For the nth throw shown on the x-axis, the corresponding performance error is plotted on the y-axis. By applying fixed-model learning, the robot system successfully performed the task after the eighth iteration.

It is important to generalize the fixed-model learning procedure so that it can be applied to other tasks. The system task commands are labeled with the vector $\mathbf{c}$ and the performance variables with the vector $\mathbf{p}$. Generalizing the error correction equation to a multi-dimensional task, we write

$$\mathbf{\Delta}_{n+1} = \mathbf{\Delta}_n - (\mathbf{p}_n - \mathbf{p}_d) \tag{8}$$

where $\mathbf{p_d}$ is the desired system performance. We also extend Equation (7) to

$$\mathbf{c}_{n+1} = \hat{\mathbf{S}}^{-1}(\mathbf{p_d} + \mathbf{\Delta}_{n+1}) \tag{9}$$

Fixed-model learning has successfully been applied to a multi-dimensional system in the case of trajectory following [Atkeson and McIntyre 1986]. A similar approach has also been developed for the task of kinematic positioning at a visual target [Atkeson et al. 1987].

The convergence of fixed-model learning depends on the input/output behavior of both the model and system. The convergence criteria can be derived by using fixed point theory [Wang 1984; Wang and Horowitz 1985]. A learning algorithm can be viewed as a mapping of error corrections on the nth attempt to error corrections on the next attempt

$$\mathbf{\Delta}_{n+1} = \mathbf{F}(\mathbf{\Delta}_n) \tag{10}$$

The fixed-model learning algorithm can be put into this form by substituting Equations (1) and (9) into Equation (8). Fixed-model learning modifies the nth error correction term by adding an amount based on the predicted performance transformed by the actual system

$$\mathbf{\Delta}_{n+1} = \mathbf{\Delta}_n - \mathbf{S}(\hat{\mathbf{S}}^{-1}(\mathbf{p_d} + \mathbf{\Delta}_n)) + \mathbf{p_d} \tag{11}$$

Note that when the correct command $\mathbf{c}^*$ is achieved by using the right error correction term $\mathbf{\Delta}^*$, then $\mathbf{c}^* = \hat{\mathbf{S}}^{-1}(\mathbf{p_d} + \mathbf{\Delta}^*)$. When the desired performance $\mathbf{p_d}$ is achieved using this correct command $\mathbf{c}^*$, then $\mathbf{p_d} = \mathbf{S}(\mathbf{c}^*)$, and Equation (11) reduces to the fixed point $\mathbf{\Delta}_{n+1} = \mathbf{\Delta}_n = \mathbf{\Delta}^*$.

We can ask whether this fixed point is stable by analyzing a linearization of Equation (11) at the point $(\mathbf{c}, \mathbf{p}) = (\mathbf{c}^*, \mathbf{p_d})$. We begin by writing two equations for small perturbations around the fixed point. For perturbations $\delta\mathbf{c}$ and $\delta\mathbf{p}$ from the fixed point,

$$\begin{aligned} \mathbf{S}(\mathbf{c}^* + \delta\mathbf{c}) &= \mathbf{p_d} + \mathbf{J}(\mathbf{c}^*)\delta\mathbf{c} && (12) \\ \hat{\mathbf{S}}^{-1}(\mathbf{p_d} + \mathbf{\Delta}^* + \delta\mathbf{\Delta}_n) &= \hat{\mathbf{S}}^{-1}(\mathbf{p_d} + \mathbf{\Delta}^*) + \hat{\mathbf{J}}^{-1}(\mathbf{p_d} + \mathbf{\Delta}^*)\delta\mathbf{\Delta}_n && (13) \end{aligned}$$

where $\mathbf{J}$ is the Jacobian matrix for the system $\mathbf{S}()$, $\hat{\mathbf{J}}$ is the Jacobian matrix for the model $\hat{\mathbf{S}}()$, and $\hat{\mathbf{J}}^{-1}$ is the Jacobian matrix for the inverse model $\hat{\mathbf{S}}^{-1}()$. To analyze the fixed point for stability, we consider the case in which the nth error correction term is perturbed from $\mathbf{\Delta}^*$ by $\delta\mathbf{\Delta}_n$ so that $\mathbf{\Delta}_n = \mathbf{\Delta}^* + \delta\mathbf{\Delta}_n$. The change in the next error correction term, $\delta\mathbf{\Delta}_{n+1} = \mathbf{\Delta}_{n+1} - \mathbf{\Delta}^*$, can be computed by substituting Equations (12) and (13) into Equation (11)

$$\delta\mathbf{\Delta}_{n+1} = (\mathbf{I} - \mathbf{J}(\mathbf{c}^*)\hat{\mathbf{J}}^{-1}(\mathbf{p_d} + \mathbf{\Delta}^*))\delta\mathbf{\Delta}_n \tag{14}$$

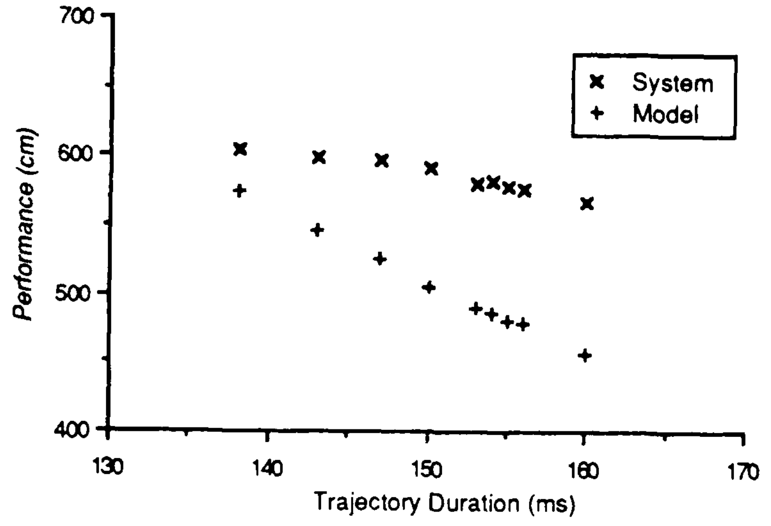

Figure 6: Command-Performance Behavior

Alternatively, we can obtain the error in the next command by multiplying through Equation (14) by $\hat{\mathbf{J}}^{-1}(\mathbf{p_d} + \mathbf{\Delta}^*)$ and noting that $\delta\mathbf{c_n} = \hat{\mathbf{J}}^{-1}(\mathbf{p_d} + \mathbf{\Delta}^*)\delta\mathbf{\Delta_n}$

$$\delta\mathbf{c_{n+1}} = (\mathbf{I} - \hat{\mathbf{J}}^{-1}(\mathbf{p_d} + \mathbf{\Delta}^*)\mathbf{J}(\mathbf{c}^*))\delta\mathbf{c_n} \tag{15}$$

The matrix $(\mathbf{I} - \hat{\mathbf{J}}^{-1}\mathbf{J})$ indicates whether fixed-model learning will converge. When the model and system are both *linear* functions of their inputs, the matrix $(\mathbf{I} - \hat{\mathbf{J}}^{-1}\mathbf{J})$ provides global convergence criteria. The command error $\delta\mathbf{c}$ will decrease when all the eigenvalues of the matrix $(\mathbf{I} - \hat{\mathbf{J}}^{-1}\mathbf{J})$ are less than one in absolute value, with the rate of decrease determined by the magnitude of the eigenvalues. If the magnitudes of all the eigenvalues are less than one, the learning process is stable and performance improves with practice. The magnitude of the eigenvalues of $(\mathbf{I} - \hat{\mathbf{J}}^{-1}\mathbf{J})$ depends on how accurately $\hat{\mathbf{J}}^{-1}$ inverts $\mathbf{J}$, and thus the stability of the learning algorithm depends on how closely the learning operator inverts the controlled system. This is a benefit of better modeling.

In the general case where model and system are *non-linear* functions of their inputs, it is difficult to develop global convergence criteria. The criteria developed for the linear case can be applied locally, however, as a necessary but not sufficient condition for convergence. Thus, all the eigenvalues of the matrix $(\mathbf{I} - \hat{\mathbf{J}}^{-1}\mathbf{J})$ must be less than one in absolute value for learning to converge. If the magnitude of any eigenvalue is greater than one, fixed-model learning will almost certainly degrade performance. The better the model approximates the system, the closer the magnitudes will be to zero, and the more likely learning is to converge.

We applied this local convergence criteria to the ball throwing system. Plots of the

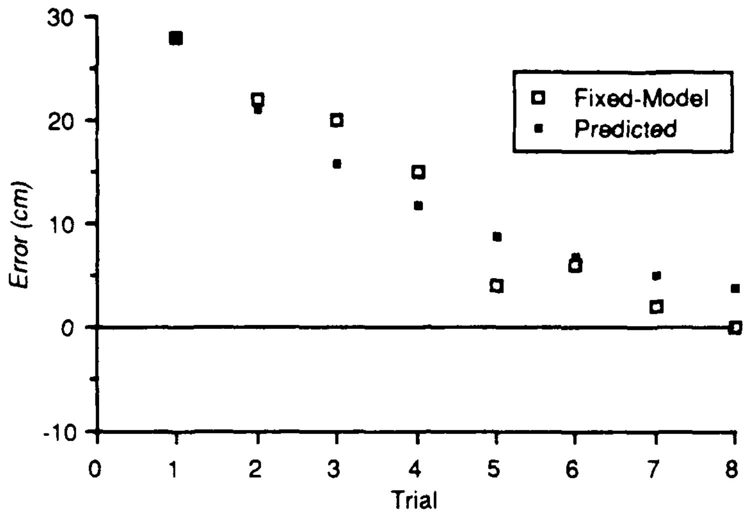

Figure 7: Convergence of Fixed-Model Learning

command–performance behavior of both the model and system are shown in Figure 6. The model and system were fit by linear functions: $\mathbf{J}$ was estimated to be -1.70 and $\hat{\mathbf{J}}$ was estimated to be -5.44. The eigenvalue of $(\mathbf{I} - \hat{\mathbf{J}}^{-1}\mathbf{J})$ for this model and system was then calculated to be 0.69. This value is less than one, indicating that the ball throws are likely to converge to the target. The open boxes in Figure 5 demonstrate that the ball throws did in fact converge.

The *performance* of the learning procedure refers to the *rate* of convergence. The geometric rate of convergence is given by the magnitudes of the eigenvalues of $(\mathbf{I} - \hat{\mathbf{J}}^{-1}\mathbf{J})$. The best performance is achieved when all the eigenvalues are close to zero. In the ball throwing task, for example, the geometric rate of convergence was calculated to be 0.69. The small squares in Figure 7 illustrate this theoretical rate of convergence. The actual iterations of fixed-model learning, denoted by open boxes, closely approximate this prediction. As a model is made more accurate, the eigenvalues of the matrix $(\mathbf{I} - \hat{\mathbf{J}}^{-1}\mathbf{J})$ approach zero leading to fast convergence and improved learning.

2.2 Refined-Model Learning

Refined-model learning refines the model as well as the command during practice. The refined-model approach essentially constructs a local linear model of the system from the last $m + 1$ attempts at the task, given a system of m inputs and m outputs. Thus, this method is an alternative to fixed-model learning only after the $(m + 1)$th iteration. Once in use, refined-model learning sacrifices the original model structure for a simpler local model. This local model is updated after each attempt at the task, leading to a refined system command.

In ball throwing, refined-model learning was applied after two attempts at the task. The first throw with a trajectory duration of 138*ms* resulted in a performance error of 28*cm*. The second throw with a 143*ms* duration resulted in a performance error of 22*cm*. Refined-model learning linearly extrapolated between these two points, suggesting a throw with a trajectory duration of 160*ms*. Given the performance error, e_n

$$e_n = \mathrm{hit}_n - \mathrm{target} \tag{16}$$

the iteration rule for refined-model learning is

$$\mathrm{duration}_{n+1} = \mathrm{duration}_n - \frac{e_n}{(e_{n-1} - e_n)/(\mathrm{duration}_{n-1} - \mathrm{duration}_n)} \tag{17}$$

The results of refined-model learning for the ball throwing task are given by the open triangles in Figure 5. The desired performance was reached in just five iterations.

Refined-model learning can be generalized to multi-dimensional tasks. We first define the performance error $\mathbf{e}_n$ to be the difference between actual and desired performance on the nth iteration

$$\mathbf{e}_n = \mathbf{p}_n - \mathbf{p}_d \tag{18}$$

We next define $\Delta\mathbf{C}$ and $\Delta\mathbf{E}$ to be $m \times m$ matrices

$$\Delta\mathbf{C} = [\ \mathbf{c}_0 - \mathbf{c}_n \quad \mathbf{c}_1 - \mathbf{c}_n \quad \cdots \quad \mathbf{c}_{n-1} - \mathbf{c}_n\ ] \tag{19}$$

$$\Delta\mathbf{E} = [\ \mathbf{e}_0 - \mathbf{e}_n \quad \mathbf{e}_1 - \mathbf{e}_n \quad \cdots \quad \mathbf{e}_{n-1} - \mathbf{e}_n\ ] \tag{20}$$

where the $\boldsymbol{\Delta}$ should not be confused with the error correction term in fixed-model learning. The general refined-model learning equation is then written as

$$\mathbf{c}_{n+1} = \mathbf{c}_n - \Delta\mathbf{C}(\Delta\mathbf{E})^{-1}\mathbf{e}_n \tag{21}$$

Refined-model learning in this form is similar to the secant method of finding zeros of functions [Gragg and Stewart 1976]. As such, improvements for avoiding singularities and for hastening convergence that are found in the numerical methods literature can be readily applied. Our primary interest here is to propose general learning procedures which can later be refined as they appear promising.

Refined-model learning will converge only if the first attempts at the task are sufficiently near the desired performance and if the system function is sufficiently smooth in that neighborhood. Because of these two restrictions, a principled and conservative approach to extrapolation should be taken. In the case of a one-dimensional system,

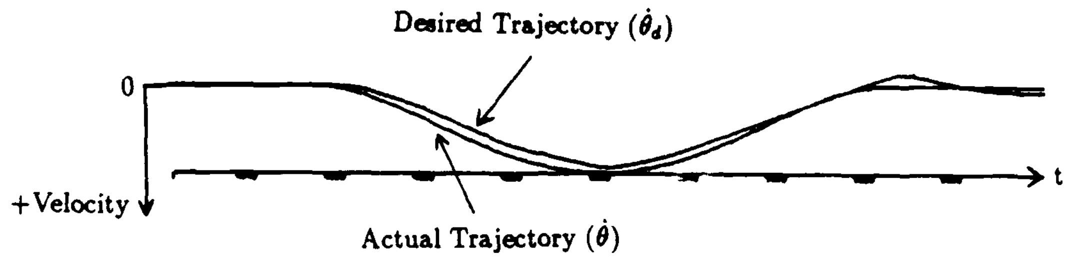

Figure 8: Actual and Desired Arm Trajectories for the Final Throw

for example, it might be useful to set a limit as to how far to believe an extrapolation. It might also be wise to interpolate instead of extrapolating as soon as a point on both sides of the desired performance is found [Press et al. 1986].

The *performance* of refined-model learning depends primarily on the input/output behavior of the system. The performance on the first learning iteration also depends on the accuracy of the internal model. A better model makes the original performance error smaller, making learning faster. In the case of a one-dimensional system, the performance error, e_n, will decay superlinearly. This convergence rate is faster than for fixed-model learning, which only converges geometrically [Forsythe et al. 1977]. Figure 5 demonstrates this performance advantage in the throwing task.

3 Discussion

The most important contribution of this work is to demonstrate that learning at the task-level can take place in the absence of perfect modeling at lower levels. Further, the top-level task is learned and not the individual modules that make up the system. In our ball-throwing robot system, the ballistic, kinematic, and dynamic models were never improved, whereas the task performance was. The desired trajectory of the robot arm was never perfectly followed, for example, but the task was accomplished. Figure 8 shows the desired velocity trajectory on the final throw that resulted in a perfect hit (zero error) as well as the actual velocity trajectory of the robot arm. The difference between the desired and actual trajectories is an indication that learning can proceed at the task level, even though lower level modules do not perform perfectly.

Many important issues remain to be investigated further. One issue is how the repeatability of the robot system affects the learning process. Noise will affect both the performance and stability of each learning algorithm differently. Sensor and actuator noise was present in our robot system when throwing $5.75m$, leading to an error range of about $0.04m$ in the target plane. As a result, it may be useful to filter the measured performance errors or the calculated command corrections at the cost of reducing the rate of convergence. Both experimental and theoretical work needs to be done on this issue.

While the system is learning a task, the model itself can be made more accurate. Refined-model learning takes this approach, as it builds a local linear model of the actual system to produce the next task command. The experience of all but the last two attempts at the task is ignored, however. Another potential approach is to build a local model on top of the structural model of the system as the task is practiced. A third approach is for the system to continually calibrate itself with respect to a task variable [Brooks et al. 1987]. With these more accurate models learning will be hastened.

Learning at the lower level can sometimes be more stable. Instead of applying task-level learning to the system, learning can be applied to each module independently. This modular learning approach can guarantee convergence in certain cases where task-level learning actually degrades performance. The convergence rate and the robustness to noise in the intermediate sensors are two issues that must be investigated, however.

One of the most striking aspects of human learning is that we generalize our experience to related tasks. While we have developed some general learning procedures that apply to many tasks, some theoretical work is necessary to suggest when practice on one task can be generalized to another.

4 Conclusions

Learning from practice has been demonstrated to improve the performance of a fairly simple task—ball throwing. Task-level performance errors are reduced by actually attempting the task. The lower level component modules of the system can be largely ignored during this learning process. The two learning procedures are both an alternative to extensive model calibration and a complement to more accurate modeling.

The performance of many different tasks can be improved with fixed-model and refined-model learning. For a given task, the commands and performance can be defined, the system can be modeled, and a learning procedure can then be applied. We plan to demonstrate that the learning procedures developed here are applicable to complex tasks such as juggling.

This research may also provide some insight into how people achieve everyday tasks. Actual implementation forces us to consider what internal models, computational resources, memory capacities, and learning procedures are necessary to achieve even the most routine tasks.

References

Arimoto, S., S. Kawamura, F. Miyazaki, and S. Tamaki, "Learning Control Theory for Dynamical Systems", Proc. 24th Conf. on Decision and Control, (Fort Lauderdale, Florida, Dec. 11-13, 1985).

Asada, H. and K. Youcef-Toumi, *Direct Drive Robots: Theory and Practice* (MIT Press, Cambridge, 1987).

Atkeson, C.G., E.W. Aboaf, J. McIntyre, and D.J. Reinkensmeyer, "Model-Based Robot Learning", Fourth Intl. Symposium on Robotics Research, (Santa Cruz, CA, August 9-14, 1987).

Atkeson, C.G., C. An, and J.M. Hollerbach, "Estimation of Inertial Parameters of Manipulator Loads and Links", *International Journal of Robotics Research* **5** (3): (1986) pp. 101-119.

Atkeson, C. G. and J. McIntyre, "Robot Trajectory Learning Through Practice", IEEE Conf. on Robotics and Automation, (San Francisco, CA, April 7 - 10, 1986).

Brooks, R.A., A.M. Flynn and T. Marill, "Self Calibration of Motion and Stereo Vision for Mobile Robots", Fourth Intl. Symposium on Robotics Research, (Santa Cruz, CA, August 9-14, 1987).

Casalino, G. and L. Gambardella, "Learning of Movements in Robotic Manipulators", Proc. IEEE Conf. on Robotics and Automation, (San Francisco, CA, April 7-10, 1986).

Craig, J. J., "Adaptive Control of Manipulators Through Repeated Trials", Proc. American Control Conference, (San Diego, June 6-8, 1984).

Forsythe, G.E., M.A. Malcolm, and C.B. Moler, *Computer Methods for Mathematical Computations* (Prentice Hall, Englewood Cliffs, 1977).

Furuta, K. and M. Yamakita, "Iterative Generation of Optimal Input of a Manipulator", Proc. IEEE Conf. on Robotics and Automation, (San Francisco, CA, April 7-10, 1986).

Gragg, W.B., and G.W. Stewart, "A Stable Variant to the Secant Method for Solving Nonlinear Equations", *SIAM J. Numerical Analysis* **13** (6): (1976) pp. 889-903.

Hara, S., T. Omata, and M. Nakano, "Synthesis of Repetitive Control Systems and its Application", Proc. 24th Conf. on Decision and Control, (Fort Lauderdale, Florida, Dec. 11-13, 1985).

Harokopos, E.G., "Optimal Learning Control of Mechanical Manipulators in Repetitive Motions", Proc. IEEE Conf. on Robotics and Automation, (San Francisco, CA, April 7-10, 1986).

Hayati, S.A., and Mirmirani, M., "Improving the Absolute Positioning Accuracy of Robot Manipulators", *J. Robotic Systems* **2** : (1985) pp. 297-413.

Mayeda, H., K. Osuka, and A. Kangawa, "A new identification method for serial manipulator arms", Preprints IFAC 9th World Congress, (Budapest, July 2-6, 1984).

Mita, T., and E. Kato, "Iterative Control and its Application to Motion Control of Robot Arm - A Direct Approach to Servo-Problems", Proc. 24th Conf. on Decision and Control, (Fort Lauderdale, Florida, Dec. 11-13, 1985).

Morita, A., "A Study of Learning Controllers For Robot Manipulators With Sparse Data", M.S. thesis, Mechanical Engineering, Massachusetts Institute of Technology (February 27, 1986).

Mukerjee, A., "Adaptation in biological sensory-motor systems: A model for robotic control", Proc., SPIE Conf. on Intelligent Robots and Computer Vision, SPIE Vol. 521, (Cambridge, November, 1984).

Neuman, C.P., and P.K. Khosla, "Identification of robot dynamics: an application of recursive estimation", Proc. 4th Yale Workshop on Applications of Adaptive Systems Theory, (New Haven, May 29-31, 1985).

Olsen, H.B., and G.A. Bekey, "Identification of parameters in models of robots with rotary joints", Proc. IEEE Conf. Robotics and Automation, (St. Louis, Mar. 25-28, 1985).

Press, W.H., B.P. Flannery, S.A. Teukolsky, and W.T. Vetterling, *Numerical Recipes* (Cambridge University Press, Cambridge, 1986).

Togai, M. and O. Yamano, "Learning Control and Its Optimality: Analysis and Is Application to Controlling Industrial Robots", Proc. IEEE Conf. on Robotics and Automation, (San Francisco, CA, April 7-10, 1986).

Togai, M., and O. Yamano, "Analysis and Design of an Optimal Learning Control Scheme for Industrial Robots: A Discrete Time Approach", Proc. 24th Conf. on Decision and Control, (Fort Lauderdale, Florida, Dec. 11-13, 1985).

Uchiyama, M., "Formation of High-Speed Motion Pattern of a Mechanical Arm by Trial", *Trans. of Society of Instrument and Control Engineers (Japan)* **19** (5): (1978) pp. 706-712.

Wang, S. H., "Computed Reference Error Adjustment Technique (CREATE) For The Control of Robot Manipulators", 22nd Annual Allerton Conf. on Communication, Control, and Computing, (October, 1984).

Wang, S. H., and I. Horowitz, "CREATE - A New Adaptive Technique", Proc. of the Nineteenth Annual Conf. on Information Sciences and Systems, (March, 1985).

Whitney, D.E., Lozinski, C.A., and J.M. Rourke, "Industrial Robot Forward Calibration Method and Results", *Trans. of Society of Instrument and Control Engineers (Japan)* **108** : (1986) pp. 1-8.

Wu, C.-H., "A Kinematic Error Model for the Design of Robot Manipulators", *Int. J. Robotics Research* **3** (1): (1984) pp. 58-67.

DISTRIBUTION:

Defense Technical Information Center

Computer Sciences Division
ONR, Code 1133

Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory, Code 5510

Dr. A.L. Slafkosky
Headquarters, U.S. Marine Corps (RD-1)

Psychological Sciences Division
ONR, Code 1142PT

Applied Research & Technology
ONR, Code 12

Dept. of the Navy
Naval Sea Systems Command
NAVSEA 90

Dr. Charles Schoman
David Taylor Naval Ship R&D Center
NSRDC 18